Nonlinear Classification

- The Perceptron
- Multiple-Layered Perceptron
- Generalized Linear Classifiers
- Kernel Functions
- SVM-Nonlinear Case
- Beyond the SVM Paradigm
- Kernel Dimension Reduction

The Perceptron (1)

- Review:
  - The perceptron cost
    - $J(\mathbf{w}') = \sum_{\mathbf{x}\in Y}(\delta_{\mathbf{x}}\mathbf{w}'^{T}\mathbf{x}')$
      - $\delta_{\mathbf{x}} = \begin{cases} -1, \mathbf{x} \in C_1 \\ +1, \mathbf{x} \in C_2 \end{cases}$
      - $\mathbf{w}' = [\mathbf{w}, w_0]^T$
  - Iterative minimizing the cost
    - $\mathbf{w}'(t+1) = \mathbf{w}'(t) - \mu(t)\sum_{\mathbf{x}\in Y}(\delta_{\mathbf{x}}\mathbf{x}')$

The step activation function

$$f(x) = \begin{cases} +1, x > 0 \\ -1, x < 0 \end{cases}$$

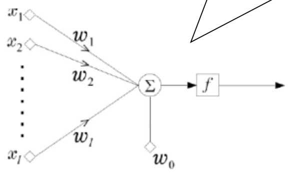


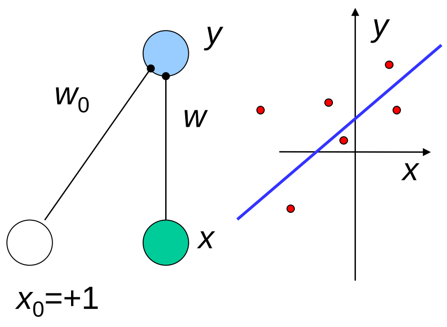
Fig. 3.5a [Theodoridis 09]

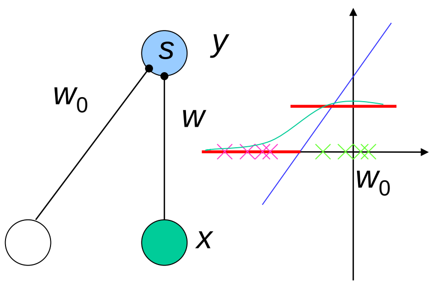
- Classification
  - $y = \mathbf{1}(\mathbf{w}^T\mathbf{x} + w_0 > 0) = f(\mathbf{w}^T\mathbf{x} + w_0)$
- Regression
  - $y = \mathbf{w}^T\mathbf{x} + w_0$

The Perceptron (2)

- Example [Alpaydin, 2020]
  - Regression
  - Classification

$y = sigmoid(\mathbf{w}^T\mathbf{x} + w_0)$

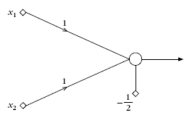




Learning Boolean Functions (1)

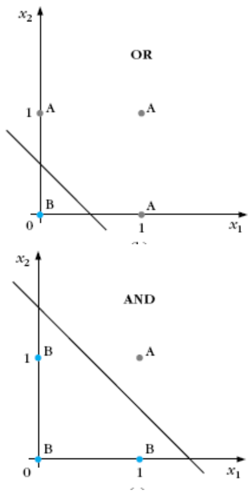
- Linearly separable
  - OR function

$$g_1(\mathbf{x}) = x_1 + x_2 - \frac{1}{2}$$



  - AND function

$$g_2(\mathbf{x}) = x_1 + x_2 - \frac{3}{2}$$



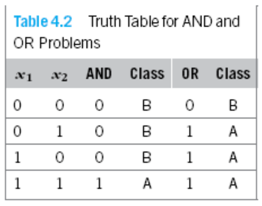


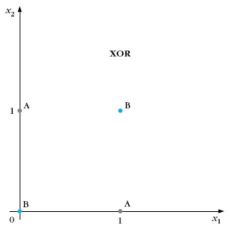
Table 4.2 Truth Table for AND and OR Problems

x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A

Figs. 4.2 & 4.3 [Theodoridis 09]

Learning Boolean Functions (2)

- XOR function
  - NOT linearly separable
  - The 1st phase
    - $y_i = f(g_i(\mathbf{x})), i = 1,2$
    - $f(x) = \begin{cases} +1, x > 0 \\ -1, x < 0 \end{cases}$
  - The 2nd phase
    - $g(\mathbf{y}) = y_1 - y_2 - \frac{1}{2}$



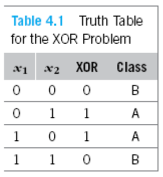
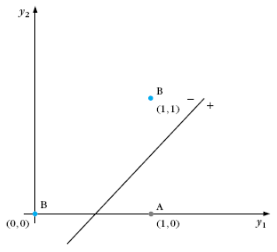
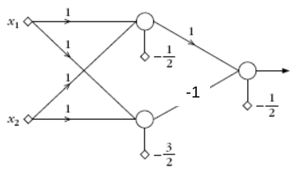


Table 4.1 Truth Table for the XOR Problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

Fig. 4.1 [Theodoridis 09]





Figs. 4.5 & 4.6 [Theodoridis 09]

Learning Boolean Functions (3)

- XOR function (cont.)
  - $g_1(\mathbf{x}) = x_1 + x_2 - \frac{1}{2}$
  - $g_2(\mathbf{x}) = x_1 + x_2 - \frac{3}{2}$
  - $\mathbf{x} \in A$, if $g_1(\mathbf{x}) > 0$ *and* $g_2(\mathbf{x}) < 0$
  - $\mathbf{x} \in B$, if $g_1(\mathbf{x}) < 0$ *or* $g_2(\mathbf{x}) > 0$

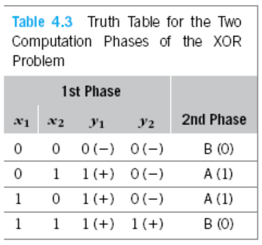


Table 4.3 Truth Table for the Two Computation Phases of the XOR Problem

1st Phase				
x_1	x_2	y_1	y_2	2nd Phase
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

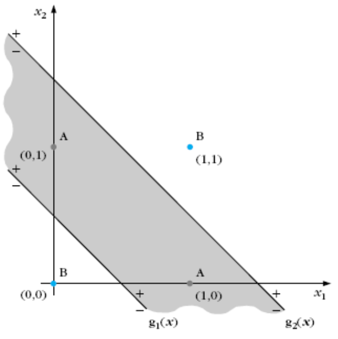


Fig. 4.4 [Theodoridis 09]

Learning Boolean Functions (4)

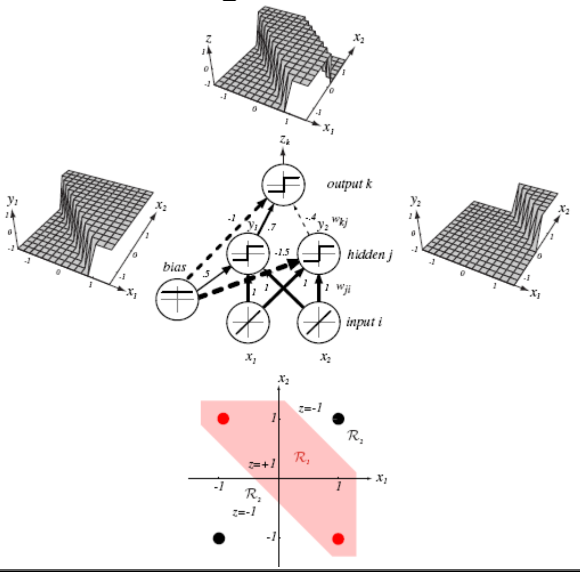
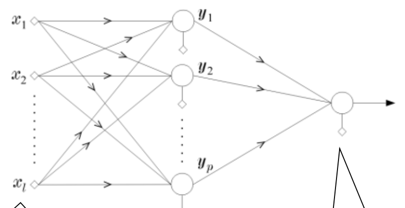


Fig. 6.1 [Duda 01]

Two-Layer Perceptron (1)

- Two-layer perceptron (or two-layer feedforward neural network)
  - Input vectors $\mathbf{x} \in R^l$
  - Hidden layer
    - p neurons (nodes)
    - $\mathbf{y} = [y_1, \dots, y_p]^T \in R^p$
      - $y_i = f(g_i(\mathbf{x}))$
      - f : nonlinear activation function

Vertices of the hypercube H_p of unit side length in p dimensional space



Input layer

Hidden layer

Output layer

Fig. 4.7 [Theodoridis 09]

Two-Layer Perceptron (2)

- Example
  - $\mathbf{x} \in R^2$
  - $g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_{i0};\ \ i = 1,2,3$

$y_i = f\left(g_i(\mathbf{x})\right) \in [0,1]$

$g(\mathbf{y}) = -y_1 - y_2 + y_3 + \frac{1}{2}$

Each neuron creates a hyperplane in x-domain

The i-th dimension corresponds to the + or - sides of g_i

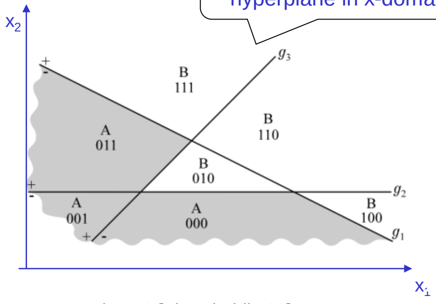


Fig. 4.8 [Theodoridis 09]

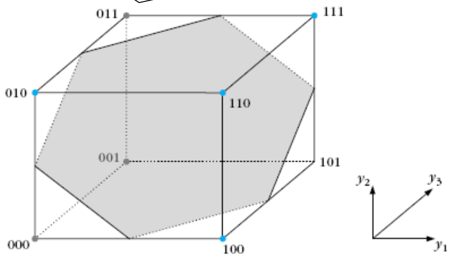


Fig. 4.9 [Theodoridis 09]

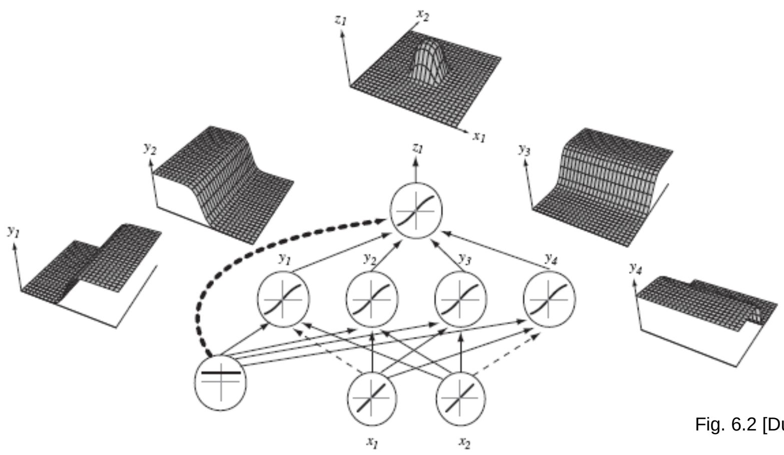


Fig. 6.2 [Duda 01]

FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units any continuous function from input to output can be approximated arbitrarily well by such a network. From Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons. Inc.

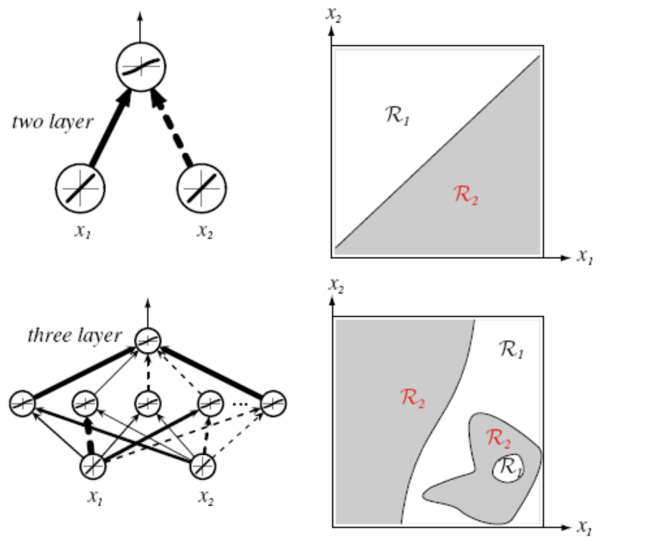


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Fig. 6.3 [Duda 01]

Activation Functions (1)

- Discrete-valued functions
  - $f(x) = \begin{cases} 1, & if\ x > 0 \\ 0, & if\ x < 0 \end{cases}$ or $f(x) = \begin{cases} +1, & if\ x > 0 \\ -1, & if\ x < 0 \end{cases}$
- Continuously differentiable functions
  - Logistic sigmoid
    - $f(x) = \sigma(ax) = \frac{1}{1+\exp(-ax)}$
      - a: the slope parameter
    - $f(-x) = 1 - f(x)$
    - $0 < f(x) < 1$
    - $f'(x) = \frac{a\exp(-ax)}{(1+\exp(-ax))^2} = af(x)(1-f(x))$
      - $f'(x) \to 0 \quad when\ f(x) \approx 0\ or\ 1$

Gradient methods can be used to solve the cost function minimization

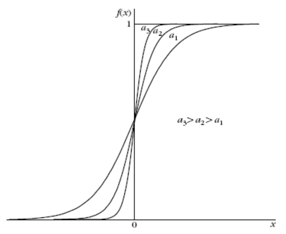


Fig. 4.12 [Theodoridis 09]

Activation Functions (2)

- Hyperbolic tangent functions
  - $f(x) = c\tanh(\frac{ax}{2}) = c\frac{1-\exp(-ax)}{1+\exp(-ax)}$
  - $f(-x) = -f(x)$

$$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)} = \frac{1-\exp(-2x)}{1+\exp(-xx)}$$
$$\tanh(-x) = -\tanh(x)$$
$$\frac{d}{dx}\tanh(x) = 1-\tanh^2(x)$$

- Rectified linear
  - $f(x) = \begin{cases} x, & if\ x > 0 \\ 0, & if\ x < 0 \end{cases}$
  - The left derivative
    - $f'(x) = \begin{cases} 1, & if\ x > 0 \\ 0, & if\ x < 0 \end{cases}$
- The leaky ReLU
  - $f(x) = \begin{cases} x, & if\ x > 0 \\ ax, & if\ x < 0 \end{cases}$

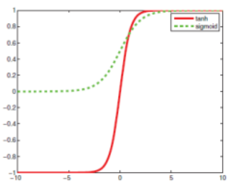


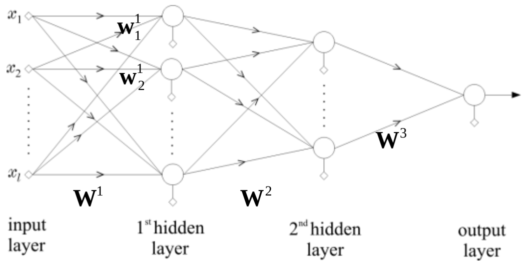
Figure 16.16 Two possible activation functions. tanh maps $\mathbb{R}$ to $[-1, +1]$ and is the preferred nonlinearity for the hidden nodes. sigm maps $\mathbb{R}$ to $[0, 1]$ and is the preferred nonlinearity for binary nodes at the output layer. Figure generated by tanhPlot.

Fig. 16.16 [Murphy]

Multiple-Layered Perceptron (1)

- Multiple-layered perceptron (MLP)
  - Also called feedforward neural network
  - L layers of neurons with k_r neurons in the rth layer

$L = 3$



Regression
$\mathbf{y}^3 = \mathbf{v}^3$
Classification
$\mathbf{y}^3 = [\dots sigm(v_i^3) \dots]$

$\mathbf{v}^1 = \mathbf{W}^1\mathbf{x}$ $\quad$ $\mathbf{v}^2 = \mathbf{W}^2\mathbf{y}^1$ $\quad$ $\mathbf{v}^3 = \mathbf{W}^3\mathbf{y}^2$

$\mathbf{y}^1 = f(\mathbf{v}^1)$ $\quad$ $\mathbf{y}^2 = f(\mathbf{v}^2)$ $\quad$ $g(\mathbf{x}) = \mathbf{y}^3 = h(\mathbf{v}^3)$

Fig. 4.10 [Theodoridis 09]

$$\boxed{\mathbf{x} \xrightarrow{\mathbf{w}^1} \mathbf{v}^1 \xrightarrow{f} \mathbf{y}^1 \xrightarrow{\mathbf{w}^2} \mathbf{v}^2 \xrightarrow{f} \mathbf{y}^2 \xrightarrow{\mathbf{w}^3} \mathbf{v}^3 \xrightarrow{h} \mathbf{y}^3}$$

Multiple-Layered Perceptron (2)

- MLP
  - The network implements a parametric non-linear classifier
  - The form of the nonlinearity is learned from the training data
- Hidden units
  - To learn nonlinear combinations of the input
    - $\mathbf{y} = f(\mathbf{v})$
      - $y_i = f(v_j)$
    - Extend classification capability
  - Activation function
    - Must be nonlinear
      - Otherwise, the whole model collapses into a large linear model
    - Does not have to be a step or a sign function
    - Is often required to be continuous and differentiable

Back-Propagation Algorithm (1)

- Assumption
  - The architecture is fixed $\mathbf{x} \xrightarrow{\mathbf{w}^1} \mathbf{v}^1 \xrightarrow{f} \mathbf{y}^1 \xrightarrow{\mathbf{w}^2} \mathbf{v}^2 \ldots \xrightarrow{\mathbf{w}^L} \mathbf{v}^L \xrightarrow{h} \mathbf{y}^L$
  - The network consists of L layers of neurons
    - With $k_0 = l$ neurons in the input layer
    - With k_r neurons in the rth layer ($r = 1, \ldots, L$)
    - All the neurons employ the same sigmoid activation function
- Goal
  - To learn the interconnection weights $\boldsymbol{\theta} = (\mathbf{W}^1, \ldots, \mathbf{W}^L)$ based on the training patterns and the desired output
    - By minimizing an appropriate cost function
      - $J(\boldsymbol{\theta}) = \sum_{i=1}^{N} E(i) = \frac{1}{2}\sum_{i=1}^{N} \|\hat{\mathbf{y}}(i) - \mathbf{y}(i)\|^2 = \frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{k_L}\left(h(v_m^L(i) - y_m(i))\right)^2$
      - $J(\boldsymbol{\theta}) = \sum_{i=1}^{N} E(i) = -\sum_{i=1}^{N}\sum_{m=1}^{k_L} y_m(i) \ln \widehat{y_m}(i)$

Back-Propagation Algorithm (2)

- The cost function
  - Highly nonlinear (depending on the weights)
  - Is usually a nonconvex function of the parameters
  - Local optimal may be found
    - Multiple restart
      - Each time using a different randomly chosen initial
    - Pick the best solution, or average over the resulting predictions
    - Comparing the resulting performance on a validation set
  - Often trained on very large data sets
    - MLPs have lots of parameters
  - Usually fits with 1st-order online methods
    - e.g., stochastic gradient descent

Back-Propagation Algorithm (3)

- Assume there are N training pairs available
  - $\{(\mathbf{x}(1), \mathbf{y}(1)), \dots, (\mathbf{x}(N), \mathbf{y}(N))\}$
    - $\mathbf{x}(i) = \begin{bmatrix} x_1(i) \\ \vdots \\ x_{k_0}(i) \end{bmatrix} \in R^{k_0} = R^l; \mathbf{y}(i) = \begin{bmatrix} y_1(i) \\ \vdots \\ y_{k_L}(i) \end{bmatrix}, \; y_k(i) = \begin{cases} 1, & \text{if } \mathbf{x} \in C_k \\ 0, & \text{otherswise} \end{cases}$
- During the training
  - When the vector $\mathbf{x}(i)$ is applied to the input
  - The output of the network is $\hat{\mathbf{y}}(i)$
  - The weights are computed such that the cost J is minimized
    - $\mathbf{w}_j^r(t+1) = \mathbf{w}_j^r(t) - \mu(t) \nabla J(\mathbf{w}_j^r(t))$
    - We need to compute $\nabla J(\mathbf{w}_j^r(t))$
      - J is dependent on $\mathbf{y}(i)$ and $\hat{\mathbf{y}}(i)$

Back-Propagation Algorithm (4)

- The cost function
  - $J(\boldsymbol{\theta}) = \sum_{i=1}^{N} E(i)$
    - $\boldsymbol{\theta} = (\mathbf{W}^1, \dots, \mathbf{W}^r)$
  - The overall gradient is obtained by summing over i
    - $\nabla_{\boldsymbol{\theta}} J = \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} E(i)$
  - Evaluating the gradient w.r.t the weight of the jth node in the rth layer
    - $\nabla_{\mathbf{w}_j^r} E(i) = \frac{\partial E(i)}{\partial v_j^r(i)} \frac{\partial v_j^r(i)}{\partial \mathbf{w}_j^r}$ (by chain rule)

      $= \frac{\partial E(i)}{\partial v_j^r(i)} \mathbf{y}^{r-1}(i)$
    - Define $\frac{\partial E(i)}{\partial v_j^r(i)} \equiv \delta_j^r(i)$

$$\begin{tikzcd} \dots \arrow[r, "f"] & \mathbf{y}^{r-1} \arrow[r, "\mathbf{W}^r"] & \mathbf{v}^r \arrow[r, "f"] & \mathbf{y}^r \arrow[r, "\mathbf{W}^{r+1}"] & \dots \end{tikzcd}$$

$$\mathbf{v}^r = \mathbf{W}^r \mathbf{y}^{r-1}$$
$$v_j^r = \left(\mathbf{w}_j^r\right)^T \mathbf{y}^{r-1}$$

Back-Propagation Algorithm (5)

- Computation of $\delta_j^r(i)$
  - When minimizing the sum of squared error
    - $E(i) = \frac{1}{2}\|\hat{\mathbf{y}}(i) - \mathbf{y}(i)\|^2$

      $= \frac{1}{2}\sum_{m=1}^{k_L}\left(\widehat{y_m}(i) - y_m(i)\right)^2 = \frac{1}{2}\sum_{m=1}^{k_L}\left(f(v_m^L(i)) - y_m(i)\right)^2$
  - In the Lth layer (output layer)
    - The dependence of $E(i)$ on $v_j^L(i)$ is explicit
      - $\delta_j^L(i) = \frac{\partial E(i)}{\partial v_j^L(i)} = \left(f(v_j^L(i)) - y_j(i)\right) f'\left(v_j^L(i)\right) = e_j(i) f'\left(v_j^L(i)\right)$
    - The weight update for the hidden-to-output weights
      - $\mathbf{w}_j^L(t+1) = \mathbf{w}_j^L(t) - \mu(t)\nabla J\left(\mathbf{w}_j^L(t)\right) = \mathbf{w}_j^L(t) - \mu(t)\sum_{i=1}^{N}\delta_j^L(i)\,\mathbf{y}^{L-1}(i)$

        $= \mathbf{w}_j^L(t) - \mu(t)\sum_{i=1}^{N}\left(f(v_j^L(i)) - y_j(i)\right) f'\left(v_j^L(i)\right)\mathbf{y}^{L-1}(i)$

Back-Propagation Algorithm (6)

- Computation of $\delta_j^r(i)$ (cont.)

$$\cdots \xrightarrow{\mathbf{w}^{r-1}} \mathbf{v}^{r-1} \xrightarrow{f} \mathbf{y}^{r-1} \xrightarrow{\mathbf{w}^{r}} \mathbf{v}^{r} \xrightarrow{f} \mathbf{y}^{r} \xrightarrow{\mathbf{w}^{r+1}} \cdots$$

  - In the layer of $L-1, L-2, \ldots, 1$
    - The value of $v_j^{r-1}(i)$ influences all $v_k^r(i), k = 1, \ldots, k_r$ of the next layer
    - Using the chain rule ($r = L-1, L-2, \ldots, 2$)
    - $\delta_j^{r-1}(i) = \frac{\partial E(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \frac{\partial E(i)}{\partial v_k^r(i)} \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}$

      $= \left(\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r\right) f'\left(v_j^{r-1}(i)\right)$
      - $\because v_k^r(i) = (\mathbf{w}_k^r)^T \mathbf{y}^{r-1} = \sum_{m=0}^{k_{r-1}} w_{km}^r y_m^{r-1}(i) = \sum_{m=0}^{k_{r-1}} w_{km}^r f\left(v_m^{r-1}(i)\right)$
      - $\therefore \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = \frac{\partial}{\partial v_j^{r-1}(i)} \left[\sum_{m=0}^{k_{r-1}} w_{km}^r f\left(v_m^{r-1}(i)\right)\right] = w_{kj}^r f'\left(v_j^{r-1}(i)\right)$
    - The weight update for the input-to-hidden & hidden-to-hidden weights
      - $\mathbf{w}_j^r(t+1) = \mathbf{w}_j^r(t) - \mu(t) \sum_{i=1}^{N} \left(\sum_{k=1}^{k_{r+1}} \delta_k^{r+1}(i) w_{kj}^{r+1}\right) f'\left(v_j^r(i)\right) \mathbf{y}^{r-1}(i)$

Back-Propagation Algorithm (7)

Fig. 6.5 [Duda 01]

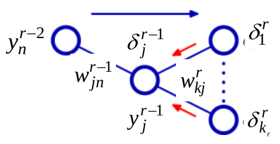
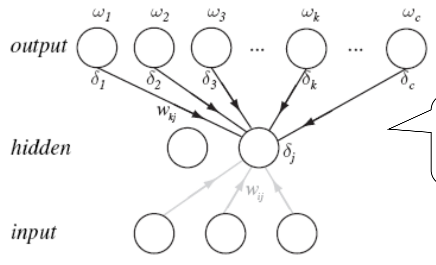


Fig. 5.7 [Bishop 06]



The layer $r-1$ error can be computed by passing the layer r errors back through $\mathbf{w}$

FIGURE 6.5. The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units: $\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k$. The output unit sensitivities are thus propagated "back" to the hidden units. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Back-Propagation Algorithm (8)

- The back-propagation algorithm
  - Initialization
    - Initialize all the weights $\mathbf{w}_j^r(i),\ j = 1, \dots, k_r;\ r = 1, \dots, L$
  - Forward computations
    - For each training vectors $\mathbf{x}(i),\ i = 1, \dots, N$
    - Compute the activations of all the hidden and output units
      - $v_j^r(i) = \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) = \sum_{k=0}^{k_{r-1}} w_{jk}^r f\left(v_k^{r-1}(i)\right)$
        - » $j = 1,2, \dots, k_r;\quad r = 1,2, \dots, L$
    - Compute the cost function
  - Backward computations
    - Compute $\delta_j^L(i) = \left(f(v_j^L(i)) - y_j(i)\right) f'\left(v_j^L(i)\right);\ j = 1, \dots, k_K;\quad \forall i$
    - Then, for $r = L-1, L-2, \dots, 2$ compute
      - $\delta_j^{r-1}(i) = f'\left(v_j^{r-1}(i)\right)\left(\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r\right);\quad j = 1,2, \dots, k_r$

Back-Propagation Algorithm (9)

- The back-propagation algorithm (cont.)
  - Update the weights
    - Evaluate the derivatives
      - $\nabla J\left(\mathbf{w}_j^r(t)\right) = \sum_{i=1}^{N} \delta_j^r(i)\, \mathbf{y}^{r-1}(i);\ \ j = 1,2, \dots, k_r;\quad r = 1,2, \dots, L$
    - Update
      - $\mathbf{w}_j^r(t+1) = \mathbf{w}_j^r(t) - \mu(t) \nabla J\left(\mathbf{w}_j^r(t)\right);\ \ j = 1,2, \dots, k_r;\quad r = 1,2, \dots, L$
- Stopping criterion
  - When the cost function or its gradient becomes smaller than a threshold
    - $J = \sum_{i=1}^{N} E(i) = \frac{1}{2} \sum_{i=1}^{N} \sum_{m=1}^{k_L} \left(f(v_m^L(i)) - y_m(i)\right)^2 < T$
    - $\|\nabla J(\mathbf{w})\| < T$

Back-Propagation Algorithm (10)

- Training protocols
  - Batch training
  - On-line training
    - Each pattern is sequentially presented (once and only once) to the algorithm
      - $\mathbf{w}_j^r(t+1) = \mathbf{w}_j^r(t) - \mu(t)\delta_j^r(i)\mathbf{y}^{r-1}(i)$
    - May help the algorithm to avoid being trapped in a local minimum
  - Stochastic training
    - The training patterns are chosen randomly from the training set and the weights are updated for each chosen pattern
  - Mini-batch gradient descent
    - To compute the gradient of a mini-batch of B data
      - special cases: $B = 1$ (SGD), $B = N$ (batch)
    - The class distribution in a mini-batch should be balanced

Back-Propagation Algorithm (11)

- Other cost functions
  - Binary classification
    - Assuming a network with a single output whose activation function is a logistic sigmoid
      - $\hat{y} = \sigma(\cdot), \quad 0 \le \hat{y} \le 1$
      - $P(y|\boldsymbol{x}) = \hat{y}^y(1-\hat{y})^{1-y}$
    - The error function = negative log-likelihood (NLL) = cross entropy
      - $J = \sum_{i=1}^N E(i) = -\sum_{i=1}^N \left(y(i)\ln\hat{y}(i) + (1-y(i))\ln(1-\hat{y}(i))\right)$
  - K separate binary classification
    - $J = \sum_{i=1}^N E(i)$
      $= -\sum_{i=1}^N \sum_{m=1}^K \left(y_m(i)\ln\widehat{y_m}(i) + (1-y_m(i))\ln(1-\widehat{y_m}(i))\right)$
  - Multi-class classification
    - $J = \sum_{i=1}^N E(i) = -\sum_{i=1}^N \sum_{m=1}^K y_m(i)\ln\widehat{y_m}(i)$

Back-Propagation Algorithm (12)

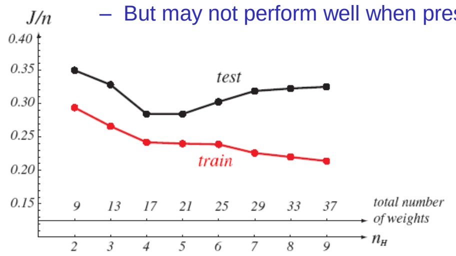
- Example 4.6 (p. 241 [Theodoridis 09])
  - Using the cross entropy cost function
    - $J = \sum_{i=1}^{N} E(i) = -\sum_{i=1}^{N} \sum_{m=1}^{K} y_m(i) \ln \widehat{y_m}(i)$
      - $\widehat{y_m}(i) = f(v_m^L(i)))$
  - and sigmoid activation function
    - $f(x) = \sigma(ax) = \frac{1}{1+\exp(-ax)} = \frac{\exp(ax)}{1+\exp(ax)}$
    - $f'(x) = \frac{a\exp(-ax)}{(1+\exp(-ax))^2} = af(x)(1-f(x))$
  - Then
    - $\delta_j^L(i) = \frac{\partial E(i)}{\partial v_j^L(i)} = \frac{\partial}{\partial v_j^L(i)}[\sum_{m=1}^{K} y_m(i) \ln f(v_m^L(i)))] = y_j(i)\frac{f'\left(v_j^L(i)\right)}{f\left(v_j^L(i)\right)}$

$$= ay_j(i)f\left(v_j^L(i)\right)\left(1-f\left(v_j^L(i)\right)\right) = ay_j(i)\hat{y}_j(i)\left(1-\hat{y}_j(i)\right)$$

Back-Propagation Algorithm (13)

- Network size
  - Too many parameters may lead to overfitting of the data
    - A large network usually results in small errors for the training set
      - Because it can learn the particular details of the training set
      - But may not perform well when presented with data unknown to it

Poor generalization



total # weights = $(2+1)n_H+n_H+1 = 4n_H+1$

FIGURE 6.15. The error per pattern for networks fully trained but differing in the numbers of hidden units, n_H. Each $2-n_H-1$ network with bias was trained with 90 two-dimensional patterns from each of two categories, sampled from a mixture of three Gaussians, and thus $n = 180$. The minimum of the test error occurs for networks in the range $4 \le n_H \le 5$, i.e., the range of weights 17 to 21. This illustrates the rule of thumb that choosing networks with roughly $n/10$ weights often gives low test error. From:

Fig. 6.15 [Duda 01]

Back-Propagation Algorithm (14)

- Overtraining
  - The network adapts to the peculiarities of the training set
- Early stopping
  - Stopping the training procedure when the error on the validation set first starts to increase

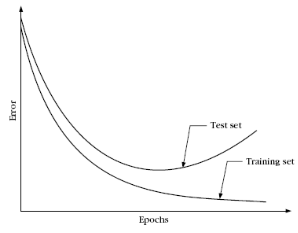


Fig. 4.14 [Theodoridis 09]

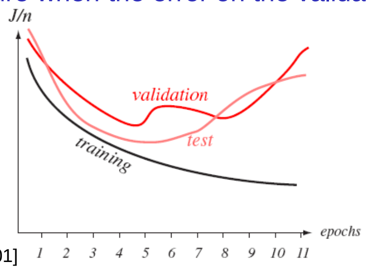


Fig. 6.6 [Duda 01]

FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation

Back-Propagation Algorithm (15)

- Learning rates

Fig. 6.16 [Duda 01]

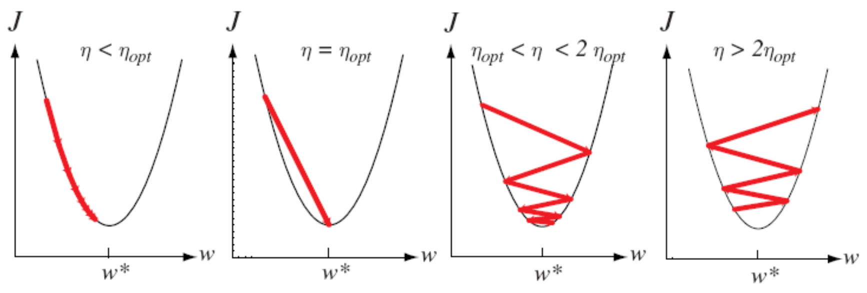


FIGURE 6.16. Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Back-Propagation Algorithm (16)

- Learning with momentum
  - To smooth out the oscillatory behavior between successive iteration steps
    - Making the weight update on the t-th iteration depend partially on the update during the $(t-1)$th iteration
    - $\mathbf{w}_j^r(t+1) = \mathbf{w}_j^r(t) + \Delta\mathbf{w}_j^r(t)$
    - $\Delta\mathbf{w}_j^r(t) = -\mu \sum_{i=1}^{N} \delta_j^r(i)\,\mathbf{y}^{r-1}(i) + \alpha\Delta\mathbf{w}_j^r(t-1), \quad 0 \le \alpha < 1$
    - or
    - $\Delta\mathbf{w}_j^r(t) = -(1-\alpha)\mu \sum_{i=1}^{N} \delta_j^r(i)\,\mathbf{y}^{r-1}(i) + \alpha\Delta\mathbf{w}_j^r(t-1),$

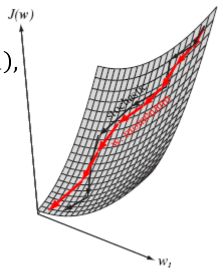


FIGURE 6.18. The incorporation of momentum into stochastic gradient descent by Eq. 37 (red arrows) reduces the variation in overall gradient directions and speeds learning. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Fig. 6.18 [Duda 01]

Back-Propagation Algorithm (17)

- Other variations
  - Using an adaptive value for the learning factor
    - $$\mu(t) = \begin{cases} r_i\mu(t-1) & \text{if } J(t) < J(t-1) \\ r_d\mu(t-1) & \text{if } J(t) > cJ(t-1) \\ \mu(t-1) & \text{if } J(t-1) \le J(t) \le cJ(t-1) \end{cases}$$
      - e.g., $r_i = 1.05,\ r_d = 0.7,\ c = 1.04$
  - Using a different learning factor for each weight
    - Increasing the learning factor if the gradient of the cost function w.r.t. the corresponding weight has the same sign on 2 successive iteration steps
    - Otherwise, reducing the learning factor
      - Because sign changes indicate a possible oscillation
  - Adopting alternative schemes for faster convergence
    - e.g., conjugate gradient, Newton, Levenberg-Marquardt algorithm

Back-Propagation Algorithm (18)

- Example (p. 301, [Duda 01])
  - Training data
    - 7 patterns from a 2-D two-class nonlinearly separable problem
  - The representations in a 2-2-1 and 2-3-1 sigmoidal network

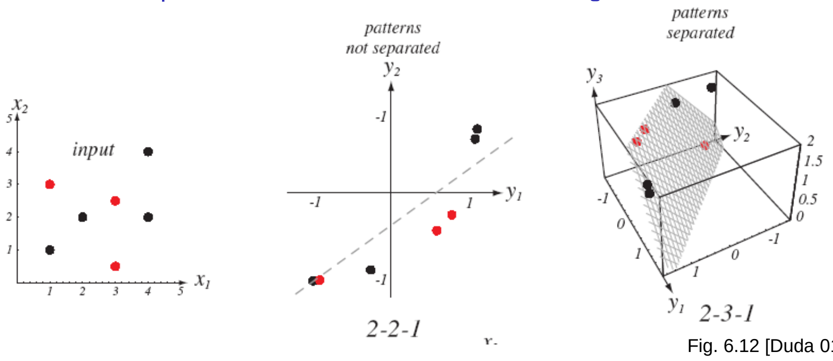


Fig. 6.12 [Duda 01]

Back-Propagation Algorithm (19)

- Example (p. 302, [Duda 01])
  - 64-2-3 sigmoidal network for classifying 3 characters (E,L,F)

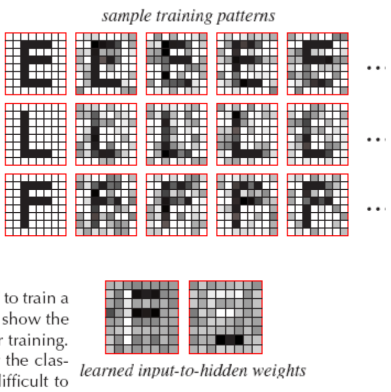


Fig. 6.13 [Duda 01]

FIGURE 6.13. The top images represent patterns from a large training set used to train a 64-2-3 sigmoidal network for classifying three characters. The bottom figures show the input-to-hidden weights, represented as patterns, at the two hidden units after training. Note that these learned weights indeed describe feature groupings useful for the classification task. In large networks, such patterns of learned weights may be difficult to interpret in this way. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Back-Propagation Algorithm (20)

- Example (p. 182 [Theodoridis 09])
  - 2 classes, each is the union of 4 Gaussians with statistically independent components ($\sigma^2 = 0.08$)
  - 400 training vectors (50 from each Gaussain)
  - 2-3-2-1 network
    - Logistic activation function with $a = 1$
    - Momentum: $\mu = 0.01, a = 0.85$
    - Adaptive momentum: $\mu = 0.01, a = 0.85, r_i = 1.05,\ r_d = 0.7,\ c = 1.05$
    - Weight initialization: U[0,1]

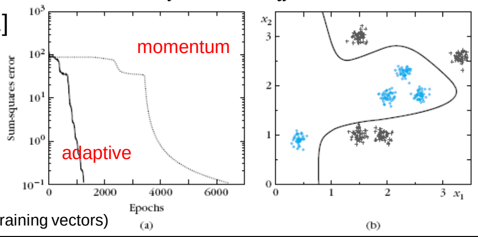


Fig. 4.15 [Theodoridis 09]

(each epoch consists of the 400 training vectors)

Back-Propagation Algorithm (21)

- Example (p. 183 [Theodoridis 09])
  - Training a sufficiently large network
    - 2-20-20-1 network, 480 weights
  - Removing the parameters that have little influence on the cost function
    - Testing the saliency values of the weights every 100 epochs
    - Removing weights with saliency value < T
    - 25 weights were left

2-20-20-1
(480 weights)

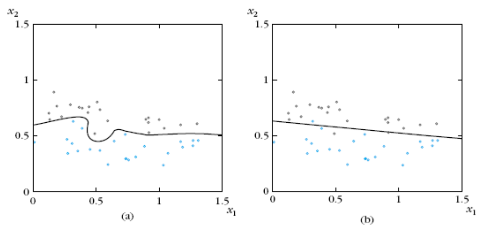


Fig. 4.16 [Theodoridis 09]

25 weights after pruning

Generalized Linear Classifiers (1)

- Generalized linear discrimination functions
  - $g(\mathbf{x}) = \sum_{i=1}^{k} w_i f_i(\mathbf{x}) = \mathbf{w}^T\mathbf{y}$
    - $\mathbf{x} \in R^l$
  - Not linear in $\mathbf{x}$, but is linear in $\mathbf{y}$
  - The functions $f_i(\mathbf{x})$ maps $\mathbf{x} \in R^l$ to $\mathbf{y} = [f_1(\mathbf{x}), \dots, f_d(\mathbf{x})]^T \in R^d$
  - The discriminant function separates points in the transformed space by a hyperplane passing through the origin

- Example 1
  - Linearly separable problem
  - $g(\mathbf{x}) = \sum_{i=1}^{l} w_i x_i + w_0 = [w_1, \dots, w_l, w_0]\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} = \mathbf{w}^T\mathbf{y}$
    - $\mathbf{w}, \mathbf{y} \in R^{l+1}$

Generalized Linear Classifiers (2)

- Example 2
  - Quadratic discriminant function ($l = 1, d = 3$)
    - $g(\mathbf{x}) = w_1 + w_2 x + w_3 x^2$

$$= [w_1, w_2, w_3]\begin{bmatrix}1\\x\\x^2\end{bmatrix} = \mathbf{w}^T\mathbf{y}$$

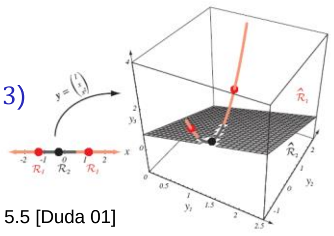


Fig. 5.5 [Duda 01]

- Example 3
  - If $\mathbf{x} \in R^2, \mathbf{y} \in R^3$
    - $g(\mathbf{x}) = \mathbf{w}^T\mathbf{y} = [w_1, w_2, w_3]\begin{bmatrix}x_1\\x_2\\\alpha x_1 x_2\end{bmatrix}$
  - The decision regions in $\mathbf{y}$ –space are convex
    - The decision surfaces in $\mathbf{x}$ –space are not convex

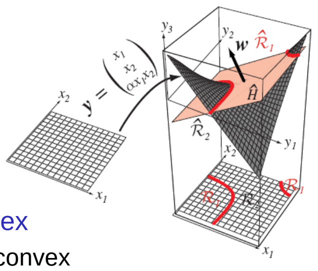


Fig. 5.6 [Duda 01]

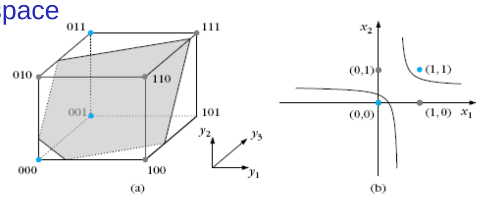
Generalized Linear Classifiers (3)

- Example 4 [Theodoridis 09]
  - The XOR function
    - $g(\mathbf{x}) = \mathbf{w}^T\mathbf{y} = [w_1, w_2, w_3]\begin{bmatrix} x_1 \\ x_2 \\ x_1x_2 \end{bmatrix}$

x_1	x_2	y_1	y_2	y_3
0	0	0	0	0
0	1	0	1	0
1	0	1	0	0
1	1	1	1	1

  - The decision hyperplane in $\mathbf{y}$ – space
    - $y_1 + y_2 - 2y_3 - \frac{1}{4} = 0$
  - The decision function in $\mathbf{x}$ – space
    - $g(\mathbf{x}) = x_1 + x_2 - 2x_1x_2 - \frac{1}{4}$

Fig. 4.20 [Theodoridis 09]



Generalized Linear Classifiers (4)

- Quadratic discriminant functions
  - Obtained by adding additional terms involving the products of pairs of components of $\mathbf{x}$
    - $g(\mathbf{x}) = w_0 + \sum_{i=1}^{l} w_i x_i + \sum_{i=1}^{l}\sum_{j=1}^{l} w_{ij}x_ix_j\,; \quad w_{ij} = w_{ji}$

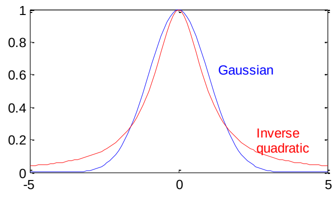
Linear part $(l+1)$ parameters

quadratic part $\frac{l(l+1)}{2}$ parameters

  - The separation surface $g(\mathbf{x}) = 0$ is a hyperquadratic surface
    - e.g., pairs of hyperplances, hyperspheres, hyperellipsoids, hyperparaboloids
  - A complete QDF involves $\frac{(l+1)(l+2)}{2}$ parameters
    - The curse of dimensionality makes it hard to capitalize on the flexibility in practice

Generalized Linear Classifiers (5)

- Radial basis functions
  - The radially symmetric basis functions
    - The output of the bases depend only on the radial distance from a center $\mathbf{c}_i$
    - Gaussian $f(\mathbf{x}) = \exp(-\frac{1}{2}\|\mathbf{x}\|^2)$
    - Inverse quadratic $f(\mathbf{x}) = \frac{1}{1+\|\mathbf{x}\|^2}$



  - The RBF model is described as
    - $g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i f_i(\frac{\|\mathbf{x}-\mathbf{c}_i\|}{\sigma_i})$
      - d: number of basis functions
      - σ: smoothing parameter
      - $\mathbf{c}_i$: centers

Kernel discriminant analysis

$$p(\mathbf{x}|C_j) \equiv \frac{1}{|X_j|}\sum_{\mathbf{x}_i \in X_j} \frac{1}{h^l}\varphi(\frac{\mathbf{x}-\mathbf{x}_i}{h})$$

$$g_j(\mathbf{x}) = \sum_{\mathbf{x}_i \in X_j} \frac{P(C_j)}{|X_j|}\varphi(\frac{\mathbf{x}-\mathbf{x}_i}{h})$$

Is a form of RBF with centers at each training data and weights determined by class priors

Generalized Linear Classifiers (6)

- Radial basis functions (cont.)
  - Equivalent to a two-layer MLP (single hidden layer) network
    - With RBF activations and linear output node

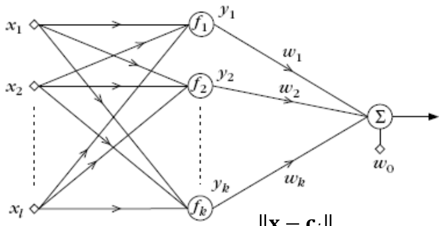


Fig. 4.17 [Theodoridis 09]

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i f_i(\frac{\|\mathbf{x}-\mathbf{c}_i\|}{\sigma_i})$$

$$y_i = f(\frac{\|\mathbf{x}-\mathbf{c}_i\|}{\sigma_i})$$

In MLP
$y_i = f\left(\mathbf{w}_i^{1^T}\mathbf{x} + w_{i0}^1\right)$

  - The output of each neuron
    - MLP: is the same for all points on a hyperplane
    - RBF: is the same for all points having the same Euclidean distance from the respective center $\mathbf{c}_i$

Generalized Linear Classifiers (7)

- Radial basis functions (cont.)
  - RBF network
    - The activation responses of the nodes are of a local nature
      - $y_i = f(\frac{\|\mathbf{x}-\mathbf{c}_i\|}{\sigma_i})$
    - Needs to use a large number of centers to fill in the space in which $g(\mathbf{x})$ is defined
  - MLP network
    - The activation responses of the nodes are of a global nature
      - $y_i = f\left(\mathbf{w}_i^{1^T}\mathbf{x} + w_{i0}^1\right)$
    - Learns slower than RBF network
    - Exhibits improved generalization properties
      - Especially for regions that are not represented sufficiently in the training set

Generalized Linear Classifiers (8)

- Radial basis functions (cont.)
  - Example (XOR problem)
    - Let $d = 2$ and $\mathbf{c}_1 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{c}_2 = \begin{bmatrix}0\\0\end{bmatrix}, f(\mathbf{x}) = \exp(-\frac{1}{2}\|\mathbf{x}\|^2), \sigma_1 = \sigma_2 = 1$
    - $\mathbf{y}(\mathbf{x}) = \begin{bmatrix} f(\frac{\|\mathbf{x}-\mathbf{c}_1\|}{\sigma_1}) \\ f(\frac{\|\mathbf{x}-\mathbf{c}_2\|}{\sigma_2}) \end{bmatrix} = \begin{bmatrix} \exp(-\frac{1}{2}\|\mathbf{x}-\mathbf{c}_1\|^2) \\ \exp(-\frac{1}{2}\|\mathbf{x}-\mathbf{c}_2\|^2) \end{bmatrix}$
    - $\begin{bmatrix}0\\0\end{bmatrix} \rightarrow \begin{bmatrix}0.135\\1\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix} \rightarrow \begin{bmatrix}1\\0.135\end{bmatrix}$
    - $\begin{bmatrix}1\\0\end{bmatrix} \rightarrow \begin{bmatrix}0.368\\0.368\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix} \rightarrow \begin{bmatrix}0.368\\0.368\end{bmatrix}$
    - $g(\mathbf{x}) = w_0 + \sum_{i=1}^{2} w_i f_i(\frac{\|\mathbf{x}-\mathbf{c}_i\|}{\sigma_i})$
      $= -1 + y_1 + y_2$

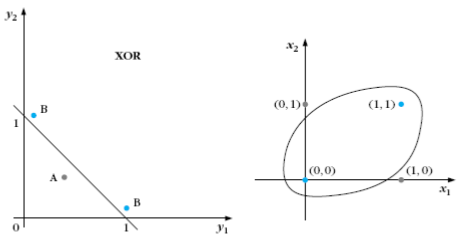


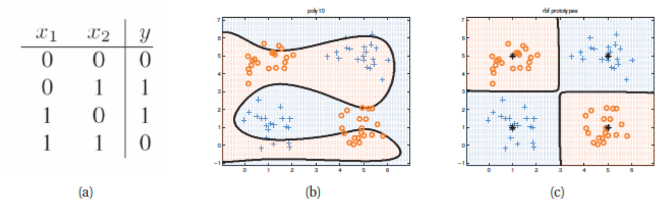
Fig. 4.21 [Theodoridis 09]

Generalized Linear Classifiers (9)

- Radial basis functions (cont.)
  - Centers
    - Random selection from data set
      - Fast but sparse regions may not be covered by RBF
    - Clustering
      - Using cluster centers as RBF centers
    - Training
      - By minimizing
      - $J = \sum_{i=1}^{N}\left(y(i) - g(\mathbf{x}(i))\right)^2 = \sum_{i=1}^{N}\left(y(i) - \left(w_0 + \sum_{j=1}^{d} w_j f_j\left(\frac{\|\mathbf{x}(j)-\mathbf{c}_j\|}{\sigma_j}\right)\right)\right)^2$

» $\begin{bmatrix} w_j(t+1) \\ \mathbf{c}_j(t+1) \\ \sigma_j(t+1) \end{bmatrix} = \begin{bmatrix} w_j(t) \\ \mathbf{c}_j(t) \\ \sigma_j(t) \end{bmatrix} - \mu(t) \begin{bmatrix} \frac{\partial J}{\partial w_j(t)} \\ \frac{\partial J}{\partial \mathbf{c}_j(t)} \\ \frac{\partial J}{\partial \sigma_j(t)} \end{bmatrix}, j = 1, \dots, d$

Computational demanding!



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 14.2 (a) xor truth table. (b) Fitting a linear logistic regression classifier using degree 10 polynomial expansion. (c) Same model, but using an RBF kernel with centroids specified by the 4 black crosses. Figure generated by `logregXorDemo`.

Fig. 14.2 [Murphy]

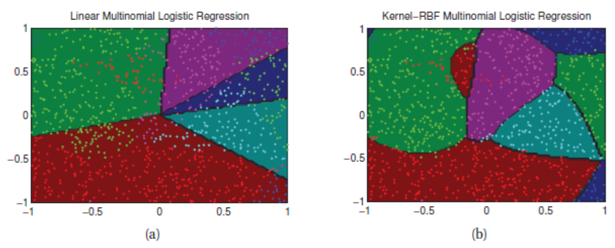


Figure 8.9 (a) Multinomial logistic regression for 5 classes in the original feature space. (b) After basis function expansion, using RBF kernels with a bandwidth of 1, and using all the data points as centers. Figure generated by `logregMultinomKernelDemo`.

Fig. 8.9 [Murphy]

SVM – Nonlinear Case (1)

- Nonlinear case
  - Each pattern $\mathbf{x}_i$ is nonlinearly transformed to $\boldsymbol{\varphi}(\mathbf{x}_i), i = 1, \dots, N$
  - The linear discriminant function in the transformed space is
    - $g(\mathbf{x}) = w^T\boldsymbol{\varphi}(\mathbf{x}) + w_0$
  - The dual form of the Lagrangian
    - $L_d(\boldsymbol{\lambda}) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\lambda_j y_i y_j \boldsymbol{\varphi}(\mathbf{x}_i)^T\boldsymbol{\varphi}(\mathbf{x}_j)$
      - y_i is the class indicator
      - $\sum_{i=1}^{N} \lambda_i y_i = 0$
      - $0 \le \lambda_i \le C; \quad i = 1, \dots, N$
  - Once λ_i are obtained via numerical quadratic programming
    - $\mathbf{w} = \sum_{i\in SV} \lambda_i y_i \boldsymbol{\varphi}(\mathbf{x}_i)$
    - $g(\mathbf{x}) = w^T\boldsymbol{\varphi}(\mathbf{x}) + w_0 = \sum_{i\in SV} \lambda_i y_i \boldsymbol{\varphi}(\mathbf{x}_i)^T\boldsymbol{\varphi}(\mathbf{x}) + w_0$

The training and the subsequent classification rely only on inner product between transformed feature vectors

Kernel Functions (1)

$$\boldsymbol{\varphi}(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \vdots \\ \sqrt{2}x_l \\ x_1^2 \\ \vdots \\ x_l^2 \\ \sqrt{2}x_1x_2 \\ \vdots \\ \sqrt{2}x_1x_l \\ \vdots \\ \sqrt{2}x_{l-1}x_l \end{bmatrix}$$

$$\langle\boldsymbol{\varphi}(\mathbf{x}), \boldsymbol{\varphi}(\mathbf{z})\rangle = 1 + 2\sum_{i=1}^{l} x_iz_i + \sum_{i=1}^{l} x_i^2z_i^2 + \sum_{i=1}^{l}\sum_{j=i+1}^{l} 2x_ix_jz_iz_j$$

Each inner product requires $d = \frac{(l+1)(l+2)}{2}$ addition and multiplication

$O(l)$ operations

$$(\langle\mathbf{x},\mathbf{z}\rangle + 1)^2 = \langle\mathbf{x},\mathbf{z}\rangle^2 + 2\langle\mathbf{x},\mathbf{z}\rangle + 1$$
$$= \left(\sum_{i=1}^{l} x_iz_i\right)^2 + 2\sum_{i=1}^{l} x_iz_i + 1$$
$$= \sum_{i=1}^{l}\sum_{j=1}^{l} 2x_ix_jz_iz_j + 2\sum_{i=1}^{l} x_iz_i + 1$$
$$= \sum_{i=1}^{l} x_i^2z_i^2 + \sum_{i=1}^{l}\sum_{j=i+1}^{l} 2x_ix_jz_iz_j + 2\sum_{i=1}^{l} x_iz_i + 1$$

Kernel Functions (2)

- Kernels for comparing documents [Ch14.2.2, Murphy]
  - Let $\mathbf{x}$ be the bag of words representation of a document
    - Each entry is the number of times of a word occurs in that document
  - The cosine similarity does not work very well, because
    - Popular words (e.g., the, and) occur in many documents
    - The similarity is artificially boosted if a certain discriminative word occurs many times in a document
  - The tf-idf kernel gives good results for information retrieval
    - $K(\mathbf{x},\mathbf{z}) = \frac{\boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\mathbf{z})}{\|\boldsymbol{\varphi}(\mathbf{x})\| \|\boldsymbol{\varphi}(\mathbf{z})\|}$, $\boldsymbol{\varphi}(\mathbf{x}) = \text{tf\_idf}(\mathbf{x})$; $\text{tf\_idf}(x_j) = \text{tf}(x_j) \times \text{idf}(j)$
    - The term frequency reduces the impact of frequent words
      - $\text{tf}(x_j) = \log(1 + x_j)$
    - The inverse document frequency
      - $\text{idf}(j) = \log \frac{\#\text{documents}}{1+\sum_{\text{all docusments}} \#(x_j>0)}$

Kernel Functions (3)

- Kernel functions
  - A real-valued function of two arguments
    - Generalizing the standard inner product in the input space
    - $K(\mathbf{x},\mathbf{z}) = \langle \boldsymbol{\varphi}(\mathbf{x}), \boldsymbol{\varphi}(\mathbf{z}) \rangle = \boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\mathbf{z})$
  - Examples
    - $\mathbf{x} \mapsto \boldsymbol{\varphi}(\mathbf{x}) = \mathbf{x}$
      - $K(\mathbf{x},\mathbf{z}) = \mathbf{x}^T \mathbf{z}$
    - $\mathbf{x} \mapsto \boldsymbol{\varphi}(\mathbf{x}) = A\mathbf{x}$
      - $K(\mathbf{x},\mathbf{z}) = \mathbf{x}^T A^T A \mathbf{z} = \mathbf{x}^T B \mathbf{z}$
    - $\mathbf{x} \mapsto \boldsymbol{\varphi}(\mathbf{x}) = (x_i x_j)_{(i,j)=(1,1),\dots,(l,l)}$
      - e.g., $\mathbf{x} = [x_1, x_2] \mapsto \boldsymbol{\varphi}(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2} x_1 x_2]$
      - $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2 = \left(\sum_{i=1}^{l} x_i z_i\right)^2 = \left(\sum_{i=1}^{l} x_i z_i\right)\left(\sum_{i=1}^{l} x_i z_i\right)$
        $= \sum_{i=1}^{l} \sum_{j=1}^{l} x_i x_j z_i z_j = \sum_{(i.j)=(1,1)}^{(l,l)} (x_i x_j)(z_i z_j)$

The inner product in the new space could be expressed as *a function in the original feature space*

Kernel Functions (4)

- Kernel function (cont.)
  - Example
    - $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z}+c)^2 = \left(\sum_{i=1}^{l} x_i z_i + c\right)\left(\sum_{i=1}^{l} x_i z_i + c\right)$
      $= \sum_{i=1}^{l}\sum_{j=1}^{l} x_i x_j z_i z_j + 2c\sum_{i=1}^{l} x_i z_i + c^2$
      $= \sum_{(i.j)=(1,1)}^{(l,l)} (x_i x_j)(z_i z_j) + \sum_{i=1}^{l}(\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2$
  - Polynomial kernels
    - The decision boundary in the input space is a polynomial curve of degree q
    - $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^q, q \geq 2$
      - $d = \binom{l+q-1}{q}$
    - $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z}+c)^q, q \geq 2$
      - $d = \binom{l+q}{q}$

Kernel Functions (5)

- Kernel function (cont.)
  - Gaussian kernel
    - $K(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{z})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{z})\right)$
    - If $\mathbf{\Sigma} = \sigma^2\mathbf{I}$, then we have a RBF kernel
      - $K(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{(\mathbf{x}-\mathbf{z})^2}{\sigma^2}\right)$
- Use of kernel functions
  - No need to
    - First create a complicated feature space
    - Conduct inner product on that space
  - Can process objects without preprocessing into feature vectors
    - Assuming there is some way of measuring the similarity between objects $K(\mathbf{x},\mathbf{z})$
      - e.g., text document, protein sequence

Kernel Functions (6)

- Properties of kernel functions
  - Symmetric
    - $K(\mathbf{x},\mathbf{z}) = \langle \boldsymbol{\varphi}(\mathbf{x}), \boldsymbol{\varphi}(\mathbf{z})\rangle = \langle \boldsymbol{\varphi}(\mathbf{z}), \boldsymbol{\varphi}(\mathbf{x})\rangle = K(\mathbf{z},\mathbf{x})$
  - Cauchy-Schwarz inequality
    - $K(\mathbf{x},\mathbf{z})^2 \le K(\mathbf{x},\mathbf{x})K(\mathbf{z},\mathbf{z})$
      - $\because |\langle \boldsymbol{\varphi}(\mathbf{x}), \boldsymbol{\varphi}(\mathbf{z})\rangle| \le \|\boldsymbol{\varphi}(\mathbf{x})\|\|\boldsymbol{\varphi}(\mathbf{z})\|$
  - The Gram matrix $\mathbf{K}$ is positive semi-definite (i.e. with non-negative eigenvalues)
    - $\mathbf{K} = \left(K(\mathbf{x}_i,\mathbf{x}_j)\right)_{N\times N} = \begin{bmatrix} K(\mathbf{x}_1,\mathbf{x}_1) & \cdots & K(\mathbf{x}_1,\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_N,\mathbf{x}_1) & \cdots & K(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix}$
      - For any finite input space $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and symmetric function $K(\mathbf{x},\mathbf{z})$

Kernel Functions (7)

- The Gram matrix $\mathbf{K}$
  - Because $\mathbf{K}$ is symmetric
    - There is an orthogonal matrix $\mathbf{V}$ that diagonalizes $\mathbf{K}$
    - $\mathbf{K} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T = \sum_{k=1}^{N} \lambda_k \mathbf{v}_k \mathbf{v}_k^T$
      - The diagonal elements of $\boldsymbol{\Lambda}$ contains the eigenvalues of $\mathbf{K}$
      - The column vectors of $\mathbf{V}$ are the corresponding eigenvectors of $\mathbf{K}$
  - Assume all eigenvalues are non-negative
    - Each element in $\mathbf{K}$ is
      - $K(\mathbf{x}_i,\mathbf{x}_j) = \sum_{k=1}^{N} \lambda_k v_{ki} v_{kj} = \left(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{V}_{:,i}\right)^T \left(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{V}_{:,j}\right)$
    - Let $\mathbf{x}_i \mapsto \boldsymbol{\varphi}(\mathbf{x}_i) = \boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{V}_{:,i} = \left[\sqrt{\lambda_1} v_{1i}, \sqrt{\lambda_2} v_{2i}, \dots, \sqrt{\lambda_N} v_{Ni}\right]^T$
    - Then we have
      - $\boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j) = \left(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{V}_{:,i}\right)^T \left(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{V}_{:,j}\right) = \sum_{k=1}^{N} \lambda_k v_{ki} v_{kj} = K(\mathbf{x}_i,\mathbf{x}_j)$

Kernel Functions (8)

- The matrix $\mathbf{K}$ is positive semi-definite
  - If we have a negative eigenvalue λ_s with eigenvector $\mathbf{v}_s$
  - Then the point $\mathbf{z}$ in the transformed space
    - $\mathbf{z} = \sum_{i=1}^{N} v_{si}\boldsymbol{\varphi}(\mathbf{x}_i) = [\boldsymbol{\varphi}(\mathbf{x}_1), \dots, \boldsymbol{\varphi}(\mathbf{x}_N)]^T\mathbf{v}_s = \sqrt{\boldsymbol{\Lambda}}\mathbf{V}^T\mathbf{v}_s$
    - Would have norm squared
      - $\|\mathbf{z}\| = \langle \mathbf{z}, \mathbf{z}\rangle = \mathbf{v}_s^T\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T\mathbf{v}_s = \mathbf{v}_s^T\mathbf{K}\mathbf{v}_s = \lambda_s < 0 \Rightarrow$ contradiction

Kernel Functions (9)

- Mercer's Theorem
  - Let $\mathbf{x} \in R^l$ and $\mathbf{x} \mapsto \boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \dots, \varphi_n(\mathbf{x}), \dots]^T$
  - Then the inner product operation has an equivalent representation
    - $\sum_{r=1}^{\infty} \varphi_r(\mathbf{x})\varphi_r(\mathbf{z}) = K(\mathbf{x}, \mathbf{z})$
    - where $K(\mathbf{x}, \mathbf{z})$ is a symmetric function satisfying
      - $\iint K(\mathbf{x}, \mathbf{z})g(\mathbf{x})g(\mathbf{z})d\mathbf{x}d\mathbf{z} \geq 0$
      - For any $g(\mathbf{x})$ such that $\int g^2(\mathbf{x})d\mathbf{x} < \infty$

The opposite is also true! Any kernel, with these properties, corresponds to an inner product in some space

The condition $\iint K(\mathbf{x}, \mathbf{z})g(\mathbf{x})g(\mathbf{z})d\mathbf{x}d\mathbf{z} \geq 0$ corresponds to the positive semi-definite condition in the finite case $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$

- Let $\mathbf{v} = [g(\mathbf{x}_1), \dots, g(\mathbf{x}_N)]^T$

$$\iint K(\mathbf{x}, \mathbf{z})g(\mathbf{x})g(\mathbf{z})d\mathbf{x}d\mathbf{z} \geq 0 \Rightarrow \sum_{i=1}^{N}\sum_{j=1}^{N} K(\mathbf{x}_i, \mathbf{x}_j)g(\mathbf{x}_i)g(\mathbf{x}_j) = \mathbf{v}^T\mathbf{K}\mathbf{v} \geq 0$$

Kernel Functions (10)

- Mercer kernels
  - Polynomial kernel
    - $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^q, q > 0$
  - Gaussian kernel
    - $K(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{z})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{z})\right)$
    - $K(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{(\mathbf{x}-\mathbf{z})^2}{\sigma^2}\right)$
  - Sigmoid kernel
    - $K(\mathbf{x},\mathbf{z}) = \tanh(\beta\mathbf{x}^T\mathbf{z} + \gamma)$

SVM – Nonlinear Case (2)

- The dual form of the Lagragian
  - $L_d(\boldsymbol{\lambda}) = \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \lambda_i\lambda_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$
    - Subject to
      - $\sum_{i=1}^N \lambda_i y_i = 0$
      - $0 \le \lambda_i \le C; \quad i = 1, \ldots, N$

We need only use K and even do not need to know φ explicitly

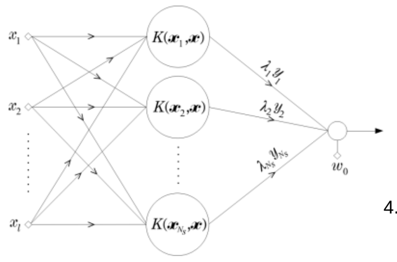
- The classifier
  - $g(\mathbf{x}) = \sum_{i\in SV} \lambda_i y_i K(\mathbf{x}_i,\mathbf{x}) + w_0$

The computational complexity is independent of the dimensionality of the kernel space
=> SVMs tend to exhibit good generalization performance

- The kernel trick is to prevent underfitting
  - To ensure that the transformed feature vector is sufficiently rich
    - So that a linear classifier can separate the data

SVM – Nonlinear Case (3)

- The SVM architecture
  - A special case of the generalized linear classifier



The number of nodes is determined by the number of support vectors N_S

4.23 [Theodoridis 09]

  - Example
    - $K(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{(\mathbf{x}-\mathbf{z})^2}{\sigma^2}\right)$
    - $\sigma = 1.75$

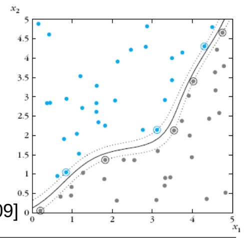


Fig. 4.24 [Theodoridis 09]

SVM – Nonlinear Case (4)

The discriminant and margins found by a polynomial kernel of degree $q = 2$

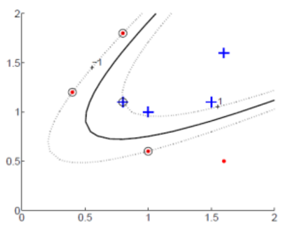


Fig. 14.4 [Alpaydin, 2020]

The discriminant and margins found by the Gaussian kernel with different σ^2

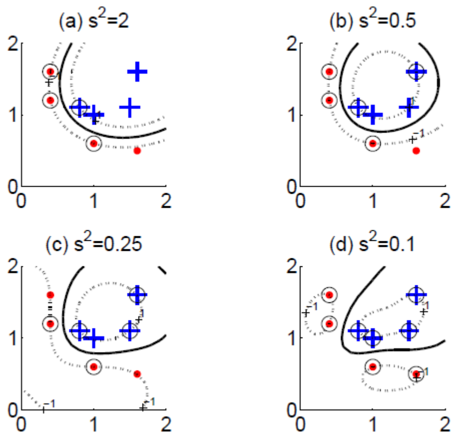


Fig. 14.5 [Alpaydin, 2020]

SVM – Nonlinear Case (5)

- If the kernel function is the RBF
  - In RBF classifier
    - The mapping to d-dimensional space is first performed
    - The centers of the RBF functions has to be determined
  - In SVM classifier
    - The number of nodes and the centers are the result of the optimization procedure
- If the kernel function is a sigmoid one
  - The SVM classifier
    - A special case of a two-layer perceptron
    - The number of nodes is the result of the optimization procedure
    - The training procedure is different from the procedure in MLP

Beyond the SVM Paradigm (1)

- Kernel trick
  - Makes the design of a linear classifier in the high-dimensional space *independent of the dimensionality* of this space
  - The designed classifier is nonlinear in the original space
- Nearest-neighbor classifier
  - Given N labeled prototypes
    - Assign $\mathbf{x}$ to the class of its nearest neighbor
    - Euclidean distance of a test vector $\mathbf{x}$ to the training points
      - $\|\mathbf{x}-\mathbf{x}_i\|^2 = \mathbf{x}^T\mathbf{x} + \mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}^T\mathbf{x}_i$
    - Kernalized NN
      - $K(\mathbf{x},\mathbf{x}) + K(\mathbf{x}_i,\mathbf{x}_i) - 2K(\mathbf{x},\mathbf{x}_i)$

Beyond the SVM Paradigm (2)

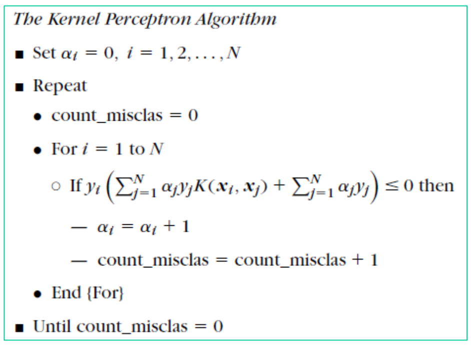
- Minimum Euclidean distance classifier
  - Feature vector $\mathbf{x}$ is assigned to the class of the nearest mean
    - $\mathbf{x} \to C_1$ if $\|\mathbf{x}-\boldsymbol{\mu}_1\|^2 < \|\mathbf{x}-\boldsymbol{\mu}_2\|^2$
      - $\boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{\mathbf{x}_i\in C_1}\mathbf{x}_i$; $\boldsymbol{\mu}_2 = \frac{1}{N_2}\sum_{\mathbf{x}_i\in C_2}\mathbf{x}_i$
    - $\|\mathbf{x}-\boldsymbol{\mu}_1\|^2 < \|\mathbf{x}-\boldsymbol{\mu}_2\|^2 \Rightarrow \mathbf{x}^T(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) < \frac{1}{2}(\|\boldsymbol{\mu}_2\|^2 - \|\boldsymbol{\mu}_1\|^2)$
    - $\Rightarrow \frac{1}{N_2}\sum_{\mathbf{x}_i\in C_2}\mathbf{x}^T\mathbf{x}_i - \frac{1}{N_1}\sum_{\mathbf{x}_i\in C_1}\mathbf{x}^T\mathbf{x}_i$
      $< \frac{1}{2}\left(\frac{1}{N_2^2}\sum_{\mathbf{x}_i\in C_2}\sum_{\mathbf{x}_j\in C_2}\mathbf{x}_i^T\mathbf{x}_j - \frac{1}{N_1^2}\sum_{\mathbf{x}_i\in C_1}\sum_{\mathbf{x}_j\in C_j}\mathbf{x}_i^T\mathbf{x}_j\right)$
  - Kernalized minimum distance classifier
    - $\mathbf{x} \to C_1$ if $\frac{1}{N_2}\sum_{\mathbf{x}_i\in C_2}K(\mathbf{x},\mathbf{x}_i) - \frac{1}{N_1}\sum_{\mathbf{x}_i\in C_1}K(\mathbf{x},\mathbf{x}_i) < T$
      - $2T = \frac{1}{N_2^2}\sum_{\mathbf{x}_i\in C_2}\sum_{\mathbf{x}_j\in C_2}K(\mathbf{x}_i,\mathbf{x}_j) - \frac{1}{N_1^2}\sum_{\mathbf{x}_i\in C_1}\sum_{\mathbf{x}_j\in C_j}K(\mathbf{x}_i,\mathbf{x}_j)$

Beyond the SVM Paradigm (3)

- Kernel perceptron algorithm
  - Single-sample correction
    - $\mathbf{w}(t+1) = \begin{cases}\mathbf{w}(t)+\mu\boldsymbol{\varphi}(\mathbf{x}_t), & \text{if } \mathbf{x}_t \in C_1, \mathbf{w}^T(t)\boldsymbol{\varphi}(\mathbf{x}_t) \le 0\\ \mathbf{w}(t)-\mu\boldsymbol{\varphi}(\mathbf{x}_t), & \text{if } \mathbf{x}_t \in C_2, \mathbf{w}^T(t)\boldsymbol{\varphi}(\mathbf{x}_t) \ge 0\\ \mathbf{w}(t), & \text{otherwise}\end{cases}$
    - Let $\mu = 1$
    - The weight update
      - $\begin{bmatrix}\mathbf{w}(t+1)\\ w_0(t+1)\end{bmatrix} = \begin{bmatrix}\mathbf{w}(t)\\ w_0(t)\end{bmatrix} + y(t)\begin{bmatrix}\boldsymbol{\varphi}(\mathbf{x}_t)\\ 1\end{bmatrix}$ if $y(t)(\mathbf{w}^T(t)\boldsymbol{\varphi}(\mathbf{x}_t) + w_0(t)) \le 0$
    - From a zero initialization
      - $\mathbf{w}(0) = \mathbf{0}$, $w_0(0) = 0$
      - The solution, after all points have been correctly classified
        - » $\mathbf{w} = \sum_{i=1}^N a_i y_i \boldsymbol{\varphi}(\mathbf{x}_i)$
        - » $w_0 = \sum_{i=1}^N a_i y_i$; a_i: counter of misclassification

Beyond the SVM Paradigm (4)

- Kernel perceptron algorithm (cont.)
  - Single-sample correction
    - The resulting nonlinear classifier
    - $g(\mathbf{x}) \equiv \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + w_0 = \sum_{i=1}^{N} a_i y_i K(\mathbf{x}_i, \mathbf{x}) + \sum_{i=1}^{N} a_i y_i$



The Kernel Perceptron Algorithm

```
■ Set $\alpha_i = 0,\ i = 1, 2, \ldots, N$
■ Repeat
  • count_misclas = 0
  • For $i$ = 1 to $N$
    ○ If $y_i \left( \sum_{j=1}^{N} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{j=1}^{N} \alpha_j y_j \right) \le 0$ then
      — $\alpha_i = \alpha_i + 1$
      — count_misclas = count_misclas + 1
  • End {For}
■ Until count_misclas = 0
```

[Theodoridis 09]

Kernel Dimension Reduction (1)

- PCA revisited
  - Given a set of centered data
    - $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}, \mathbf{x}_i \in R^d; \sum_{i=1}^{N} \mathbf{x}_i = 0$
  - PCA diagonalizes the covariance matrix
    - $\boldsymbol{\Sigma}_{\mathbf{x}} = E\{\mathbf{x}\mathbf{x}^T\} \approx \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T$
    - By solving the eigendecomposotion
      - $\boldsymbol{\Sigma}_{\mathbf{x}} \mathbf{v} = \lambda \mathbf{v}$
    - All solutions $\mathbf{v}$ (the eigenvectors of the correlation matrix) must lie in the span of the centered data $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$
      - $\because \lambda \mathbf{v} = \boldsymbol{\Sigma}_{\mathbf{x}} \mathbf{v} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i^T \mathbf{v}) \mathbf{x}_i$
    - The projection of $\mathbf{x}$ onto the subspace spanned by the eigenvectors
      - $y(k) \equiv \langle \mathbf{v}_k, \mathbf{x} \rangle, k = 1, \ldots, d$

Kernel Dimension Reduction (2)

- Kernel PCA
  - To generalize PCA by nonlinearly mapping the input data
  - $\mathbf{x} \in R^d \mapsto \boldsymbol{\varphi}(\mathbf{x}) \in H$
    - Assume the data are centered $\sum_{i=1}^{N} \boldsymbol{\varphi}(\mathbf{x}_i) = 0$
    - The correlation matrix in H
      - $\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\varphi}(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}_i)^T$
    - The goal is to solve the eigendecomposotion
      - $\boldsymbol{\Sigma}\mathbf{v} = \lambda\mathbf{v}$
    - All solutions $\mathbf{v}$ must lie in the span of $\{\boldsymbol{\varphi}(\mathbf{x}_1), \dots, \boldsymbol{\varphi}(\mathbf{x}_N)\}$
      - $\because \lambda\mathbf{v} = \boldsymbol{\Sigma}\mathbf{v} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{\varphi}(\mathbf{x}_i)^T\mathbf{v})\boldsymbol{\varphi}(\mathbf{x}_i)$
    - For $\lambda \neq 0$, there exist coefficients $a(i)$ such that
      - $\mathbf{v} = \sum_{i=1}^{N} a(i)\boldsymbol{\varphi}(\mathbf{x}_i)$

Kernel Dimension Reduction (3)

- Kernel PCA (cont.)
  - $\boldsymbol{\Sigma}\mathbf{v} = \lambda\mathbf{v}$

$$\Rightarrow \langle \boldsymbol{\varphi}(\mathbf{x}_k), \boldsymbol{\Sigma}\mathbf{v}\rangle = \lambda\langle \boldsymbol{\varphi}(\mathbf{x}_k), \boldsymbol{\Sigma}\mathbf{v}\rangle, k = 1, \dots, N$$

$$\Rightarrow \frac{1}{N}\sum_{i=1}^{N} a(i) \sum_{j=1}^{N} \langle \boldsymbol{\varphi}(\mathbf{x}_i), \boldsymbol{\varphi}(\mathbf{x}_j)\rangle\langle \boldsymbol{\varphi}(\mathbf{x}_j), \boldsymbol{\varphi}(\mathbf{x}_k)\rangle = \lambda \sum_{i=1}^{N} a(i)\langle \boldsymbol{\varphi}(\mathbf{x}_k), \boldsymbol{\varphi}(\mathbf{x}_i)\rangle$$

  - Defining an $N \times N$ matrix $\mathbf{K}$ by
    - $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\varphi}(\mathbf{x}_i), \boldsymbol{\varphi}(\mathbf{x}_j)\rangle$
      - $K(\cdot,\cdot)$: the kernel function
  - Then
    - $\frac{1}{N}\mathbf{K}^2\mathbf{a} = \lambda\mathbf{K}\mathbf{a} \Rightarrow \mathbf{K}\mathbf{a} = N\lambda\mathbf{a}$
      - $\because$ $\mathbf{K}$ has a set of eigenvectors which spans the whole space
      - $\mathbf{a} = [a\mathbf{v}, \dots, a(N)]^T$

Kernel Dimension Reduction (4)

- Kernel PCA (cont.)
  - To diagonalize $\mathbf{K}$
    - $\mathbf{Ka} = N\lambda\mathbf{a}$
    - The eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$
      - λ_p is the smallest nonzero eigenvalue
    - The corresponding eigenvectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p$
  - The kth eigenvector $\mathbf{v}_k$ of $\boldsymbol{\Sigma}$ corresponds to the kth eigenvector $\mathbf{a}_k$ of $\mathbf{K}$
    - $\mathbf{v}_k = \sum_{i=1}^{N} a_k(i)\boldsymbol{\varphi}(\mathbf{x}_i)\,; \quad k = 1, \ldots, p$
    - We need $\mathbf{v}_k$ to have unit length
      - $\mathbf{v}_k^T\mathbf{v}_k = 1 = \sum_{i=1}^{N} a_k(i)\sum_{j=1}^{N} a_k(j)\langle\boldsymbol{\varphi}(\mathbf{x}_i), \boldsymbol{\varphi}(\mathbf{x}_j)\rangle = \langle\mathbf{Ka}_k, \mathbf{a}_k\rangle = \langle N\lambda\mathbf{a}_k, \mathbf{a}_k\rangle$
      - $\mathbf{a}_k$ must be normalized to
        - » $N\lambda\mathbf{a}_k^T\mathbf{a}_k = 1\,; \quad k = 1, \ldots, p$

Kernel Dimension Reduction (5)

- Kernel PCA (cont.)
  - The basic steps
    - Compute $\mathbf{K}$
      - $K_{ij} = \langle\boldsymbol{\varphi}(\mathbf{x}_i), \boldsymbol{\varphi}(\mathbf{x}_j)\rangle = K(\mathbf{x}_i, \mathbf{x}_j); \quad i, j = 1, \ldots, N$
    - Compute the l $(l \leq p)$ dominant eigenvalues and eigenvectors $\mathbf{a}_k$ of $\mathbf{K}$
      - $\mathbf{Ka} = N\lambda\mathbf{a}$
    - Normalize the eigenvectors so that
      - $N\lambda\mathbf{a}_k^T\mathbf{a}_k = 1$
    - Compute the projection onto the dominant eigenvectors
      - $y(k) \equiv \langle\mathbf{v}_k, \boldsymbol{\varphi}(\mathbf{x})\rangle = \sum_{i=1}^{N} a_k(i)K(\mathbf{x}_i, \mathbf{x})\,; k = 1, \ldots, l$
  - Unlike linear PCA, the dominant eigenvectors $\mathbf{v}_k$ of $\boldsymbol{\Sigma}$ are NOT computed explicitly!
    - All we know are the respective nonlinear projections $\langle\mathbf{v}_k, \boldsymbol{\varphi}(\mathbf{x})\rangle$

Kernel Dimension Reduction (6)

- Kernel PCA (cont.)

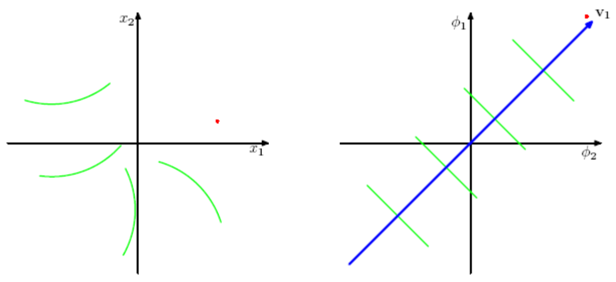


Fig. 12.16
[Bishop,06]

Figure 12.16 Schematic illustration of kernel PCA. A data set in the original data space (left-hand plot) is projected by a nonlinear transformation $\phi(\mathbf{x})$ into a feature space (right-hand plot). By performing PCA in the feature space, we obtain the principal components, of which the first is shown in blue and is denoted by the vector $\mathbf{v}_1$. The green lines in feature space indicate the linear projections onto the first principal component, which correspond to nonlinear projections in the original data space. Note that in general it is not possible to represent the nonlinear principal component by a vector in $\mathbf{x}$ space.

Kernel Dimension Reduction (7)

- Kernel LDA or Generalized DA (KDA or GDA)
  - To generalize LDA by nonlinearly mapping the input data
  - $\mathbf{x} \in R^d \mapsto \boldsymbol{\varphi}(\mathbf{x}) \in H$
    - The scatter matrices
      - $\mathbf{S}_W^{\boldsymbol{\varphi}} = \frac{1}{N}\sum_{i=1}^{K}\sum_{\mathbf{x}\in C_i}\left(\boldsymbol{\varphi}(\mathbf{x}) - \boldsymbol{\mu}_i^{\boldsymbol{\varphi}}\right)\left(\boldsymbol{\varphi}(\mathbf{x}) - \boldsymbol{\mu}_i^{\boldsymbol{\varphi}}\right)^T$
      - $\mathbf{S}_B^{\boldsymbol{\varphi}} = \sum_{i=1}^{K} P_i\left(\boldsymbol{\mu}_i^{\boldsymbol{\varphi}} - \boldsymbol{\mu}^{\boldsymbol{\varphi}}\right)\left(\boldsymbol{\mu}_i^{\boldsymbol{\varphi}} - \boldsymbol{\mu}^{\boldsymbol{\varphi}}\right)^T$
      - $\mathbf{S}_M^{\boldsymbol{\varphi}} = \mathbf{S}_W^{\boldsymbol{\varphi}} + \mathbf{S}_B^{\boldsymbol{\varphi}}$
  - The goal is to seek the optimal projective function by solving
    - $\mathbf{v}_{opt} = \underset{\mathbf{v}}{\operatorname{argmax}} \frac{\mathbf{v}^T\mathbf{S}_B^{\boldsymbol{\varphi}}\mathbf{v}}{\mathbf{v}^T\mathbf{S}_W^{\boldsymbol{\varphi}}\mathbf{v}} = \underset{\mathbf{v}}{\operatorname{argmax}} \frac{\mathbf{v}^T\mathbf{S}_B^{\boldsymbol{\varphi}}\mathbf{v}}{\mathbf{v}^T\mathbf{S}_M^{\boldsymbol{\varphi}}\mathbf{v}}$
      - $\mathbf{v}_{opt}$ is the eigenvector of $\mathbf{S}_B^{\boldsymbol{\varphi}}\mathbf{v} = \lambda\mathbf{S}_M^{\boldsymbol{\varphi}}\mathbf{v}$ associated to the largest eigenvalue λ

Kernel Dimension Reduction (8)

- KDA (cont.)
  - The solution $\mathbf{v}$ must lie in the span of $\{\boldsymbol{\varphi}(\mathbf{x}_1), \dots, \boldsymbol{\varphi}(\mathbf{x}_N)\}$
    - There exist coefficients $a(i)$ such that $\mathbf{v} = \sum_{i=1}^{N} a(i)\boldsymbol{\varphi}(\mathbf{x}_i)$
  - Because
    - $[\boldsymbol{\varphi}(\mathbf{x}_1), \dots, \boldsymbol{\varphi}(\mathbf{x}_N)]^T \mathbf{S}_B^{\boldsymbol{\varphi}} \mathbf{v} = \cdots = \frac{1}{N}\mathbf{KWKa}$
    - $\lambda[\boldsymbol{\varphi}(\mathbf{x}_1), \dots, \boldsymbol{\varphi}(\mathbf{x}_N)]^T \mathbf{S}_M^{\boldsymbol{\varphi}} \mathbf{v} = \cdots = \frac{\lambda}{N}\mathbf{KKa}$
      - $\mathbf{K}$ is an $N \times N$ kernel matrix defined as $K_{ij} = \langle \boldsymbol{\varphi}(\mathbf{x}_i), \boldsymbol{\varphi}(\mathbf{x}_j) \rangle$
      - $\mathbf{W}$ is an $N \times N$ matrix defined as
        - » $W_{ij} = \begin{cases} \frac{1}{n_k}, \text{ if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ belong to the } k\text{th class} \\ 0, \text{ otherwise} \end{cases}$
    - $\mathbf{v}_{opt} = \underset{\mathbf{v}}{\operatorname{argmax}} \frac{\mathbf{v}^T \mathbf{S}_B^{\boldsymbol{\varphi}} \mathbf{v}}{\mathbf{v}^T \mathbf{S}_M^{\boldsymbol{\varphi}} \mathbf{v}}$ is equivalent to $\mathbf{a}_{opt} = \underset{\mathbf{v}}{\operatorname{argmax}} \frac{\mathbf{a}^T\mathbf{KWKa}}{\mathbf{a}^T\mathbf{KKa}}$

Kernel Dimension Reduction (9)

- KDA (cont.)
  - The basic steps
    - Compute $\mathbf{K}$ and $\mathbf{W}$ matrices
    - Compute the eigen-decomposition of $\mathbf{K}$
      - $\mathbf{K} = \mathbf{U\Lambda U}^T$
    - Let $\boldsymbol{\beta} = \mathbf{\Sigma U}^T\mathbf{a}$
      - Compute eigenvalue and eigenvector $\boldsymbol{\beta}_k$ of
      - $\mathbf{U}^T\mathbf{WU}\boldsymbol{\beta} = \lambda\boldsymbol{\beta}$
    - Use $\mathbf{a} = \mathbf{U\Sigma}^{-1}\boldsymbol{\beta}$ to compute $\mathbf{v}$
      - $\mathbf{v} = \sum_{i=1}^{N} a(i)\boldsymbol{\varphi}(\mathbf{x}_i)$
    - Normalize the eigenvectors so that
      - $\mathbf{v}^T\mathbf{v} = \mathbf{a}^T\mathbf{Ka} = 1$
    - Compute the projection onto the eigenvectors $\mathbf{v}_k$
      - $y(k) \equiv \langle \mathbf{v}_k, \boldsymbol{\varphi}(\mathbf{x}) \rangle = \sum_{i=1}^{N} a_k(i)K(\mathbf{x}_i, \mathbf{x})\,;\, k = 1, \dots, l$

Because $\frac{\mathbf{a}^T\mathbf{KWKa}}{\mathbf{a}^T\mathbf{KKa}} = \frac{\boldsymbol{\beta}^T\mathbf{U}^T\mathbf{WU}\boldsymbol{\beta}}{\boldsymbol{\beta}^T\boldsymbol{\beta}}$